Tópico 04 - Memória cache

Hugo Vinícius Leão e Silva

hugovlsilva@gmail.com, hugo.vinicius.16@gmail.com, hugovinicius@ifg.edu.br

Instituto Federal de Educação, Ciência e Tecnologia de Goiás
Campus Anápolis
Curso de Bacharelado em Ciência da Computação

15 de maio de 2023

Visão geral

Definições preliminares

As memórias são definidas por:

- **Localidade**: se é acessível diretamente pelo processador – memória RAM, a **cache** e os registradores – ou por meio de controladores de E/S;
- **Capacidade**: qual a capacidade de armazenamento da memória em palavras ou, como é normalmente, em bytes;
- **Unidade de transferência**: quantos bytes são transferidos de e para a memória de uma só vez. Pode ser uma **unidade endereçável (UE)**[1] ou um **bloco** muito maior do que uma UE;
- **Tecnologia**: memória semicondutora, magnética, óptica, magneto-óptica etc;
- **Características físicas do armazenamento de dados**: se a memória é ou não volátil.

[1] Unidade endereçável é a menor unidade que possui endereços de memória. Pode ser uma **palavra de máquina** ou **um byte**.

Definições preliminares (cont'd)

As memórias são definidas por:

- **Tipo de acesso**: como os dados podem ser acessados:
  - **Acesso sequencial**: memória acessada **linearmente** e dividida em registros. Dados gravados na forma `endereço` `linear-`$payload$. Apenas **um** mecanismo de leitura-gravação utilizado que se mover para o registro desejado. **Tempo de acesso** altamente variável. Ex.: K7 e VHS;
  - **Acesso direto**: blocos/registros possuem um **endereço físico** exclusivo. Mecanismo de leitura-gravação **único** $\rightarrow$ tempo de acesso variável. Ex.: HD;
  - **Acesso aleatório**: unidades endereçáveis possuem um mecanismo de leitura-gravação **exclusivo** $\rightarrow$ tempo de acesso independente da sequência de acessos e é constante. Ex.: RAM e memória cache;
  - **Associativo**: memória de acesso aleatório que compara todas as suas palavras **simultaneamente**. Uma palavra é buscada de acordo com o seu **conteúdo** e não o seu endereço. Tempo de acesso constante. Ex.: memória cache;

Definições preliminares (cont'd)

As memórias são avaliadas por:

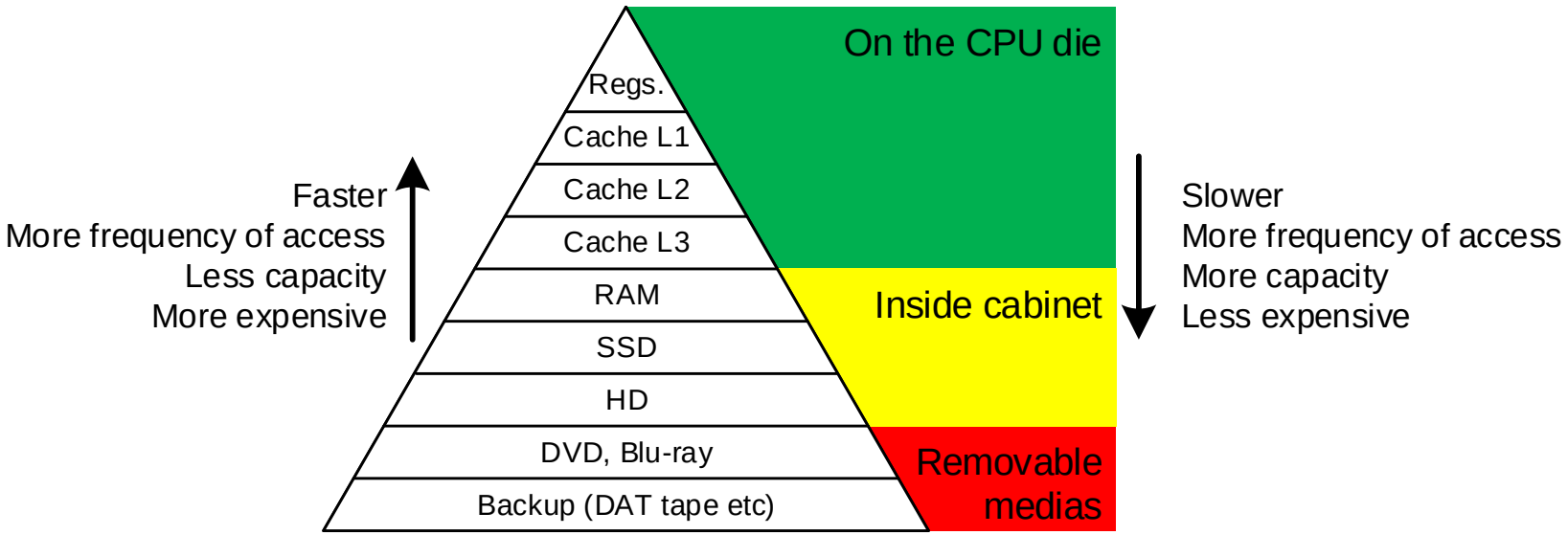
- **Capacidade de armazenamento**;
- **Desempenho**:
  - **Tempo de acesso/resposta (latência)**: tempo **completo** necessário para realizar uma operação de leitura/escrita;
  - **Tempo de ciclo**: tempo adicional (além da latência) entre duas operações consecutivas. Exemplo: tempo para dar uma "limpada" no barramento;
  - **Taxa de transferência**: taxa em que os dados são transferidos de ou para a memória.

Hierarquia de memória

- Um projetista de um computador deseja a **maior quantidade** da memória **mais rápida possível** a um **custo razoável**;
- Ele deseja que os programas e os *datasets* caibam na memória, que CPU **não** fique ociosa por muito tempo esperando pelos dados e instruções, nem que o computador seja caríssimo.
- Mas há as seguintes relações:
  - Mais rápida? Maior custo por bit.
  - Maior capacidade? Menor custo por bit, mas é mais lenta...

Hierarquia de memória

A alternativa é não usar apenas um tipo de memória, mas vários em um esquema de **hierarquia de memória**



Hierarquia de memória

- Em uma hierarquia de memória, um tipo de memória complementa a outra;
- Ex.: considere um PC com memória cache de 16 KB e tempo de acesso de 2 ns e uma memória RAM de 1 GB e tempo de acesso de 60 ns. Suponha que 95% dos acessos à memória esteja em cache. Qual o tempo médio para acessar uma palavra?

$$T_M = 0{,}95 \times (2\,\text{ns}) + 0{,}05 \times (2\,\text{ns} + 60\,\text{ns})$$

$$T_M = 0{,}95 \times (2 \times 10^{-9}) + 0{,}05 \times [(2 \times 10^{-9}) + (60 \times 10^{-9})]$$

$$T_M = 5 \times 10^{-9}$$

$$T_M = 5\text{ns}$$

- Mas, dentre outros, um requisito é que **deve haver uma hierarquia na frequência de acesso à memória pela CPU**.

Hierarquia de memória

- Em outras palavras: o princípio da **localidade de referência**[2] afirma que: durante a execução de um programa, os acessos de memória pela CPU para dados e instruções tendem a se agrupar;
- Em um pequeno período, o programa ...
  - executa um único bloco de instruções: `for`, `while`, `do...while`, `function`;
  - opera sobre o mesmo grupo de dados: vetores, matrizes ou outro conjunto de palavras **agrupadas**.

[2] Também podemos chamá-la de localidade temporal ou localidade espacial.

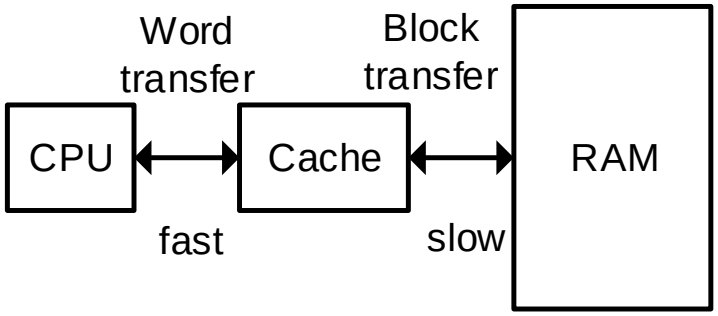
Hierarquia de memória

- Não pense em cache apenas como um *hardware*!
- O Sistema Operacional usa parte da RAM como *buffer*[3] para armazenar temporariamente dados a serem gravados em memória externa;
- Pode melhorar o desempenho do computador ao **agrupar** as gravações em disco. Poucas gravações em blocos maiores são mais rápidas do que muitas gravações pequenas;
- Dados em *buffer* podem ser usados e não precisam ser lidos novamente do disco.

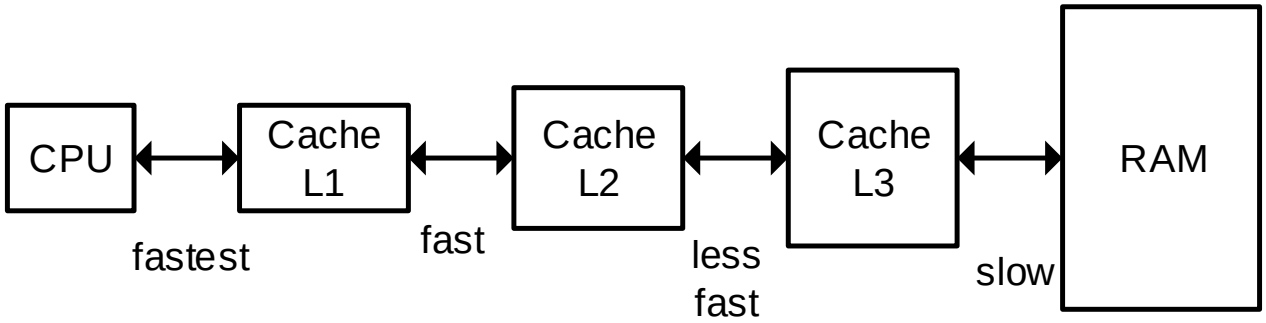
[3] Às vezes chamada de *cache* de disco.

Princípios da memória cache

A memória cache se situa entre a CPU e a memória RAM:



E podem ser usados um ou mais níveis de cache, formando uma **hierarquia** de cache:



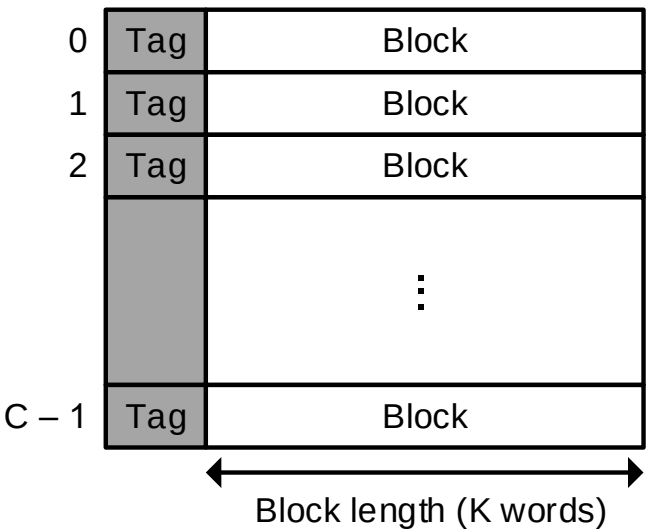
Princípios da memória cache

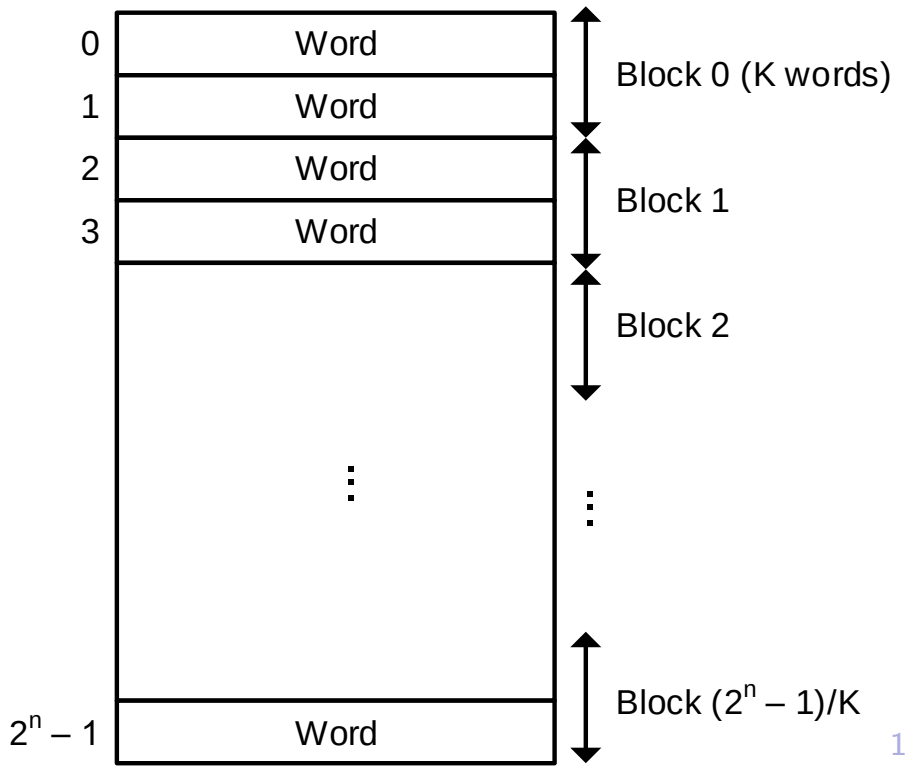
- A memória cache tem uma cópia de parte do conteúdo da memória RAM;
- Quando a CPU faz uma leitura uma **palavra** na memória RAM, a cache verifica se ela já a possui;
- Se tiver, entrega a **palavra** à CPU (***cache hit***). Senão, busca um **bloco** da memória RAM[4] e entrega a palavra à CPU (***cache miss***).

[4] Esse bloco contém a palavra solicitada e $(K-1)$ palavras vizinhas

Princípios da memória cache

- A memória RAM de $L = (2^N)$ UEs, onde $N \rightarrow$ tam. end. memória em bits, é dividida em $M = (L/K)$ **blocos** de K palavras;
- A memória cache possui $C \ll M$ **linhas** formadas por bits de controle, uma *tag* e um bloco;
- A tag identifica o bloco. Usualmente é uma parte do endereço de memória armazenado naquela linha.





Projeto de memória cache

- Praticamente todo processador "comum" e alguns processadores embarcados suportam **memória virtual**;
- As instruções utilizam endereços virtuais/lógicos em vez de endereços físicos da RAM;
- O MMU (*Memory Management Unit*) traduz endereços virtuais em físicos;
- Então, pode existir um **cache virtual** (antes da MMU) ou "**cache físico**" (depois da MMU);
- Cache virtual é mais rápido que o físico, mas os endereços físicos são diferentes para cada processo (programa) em execução. Cada mudança de contexto exige esvaziar (*flush*) o cache virtual.

Projeto de memória cache

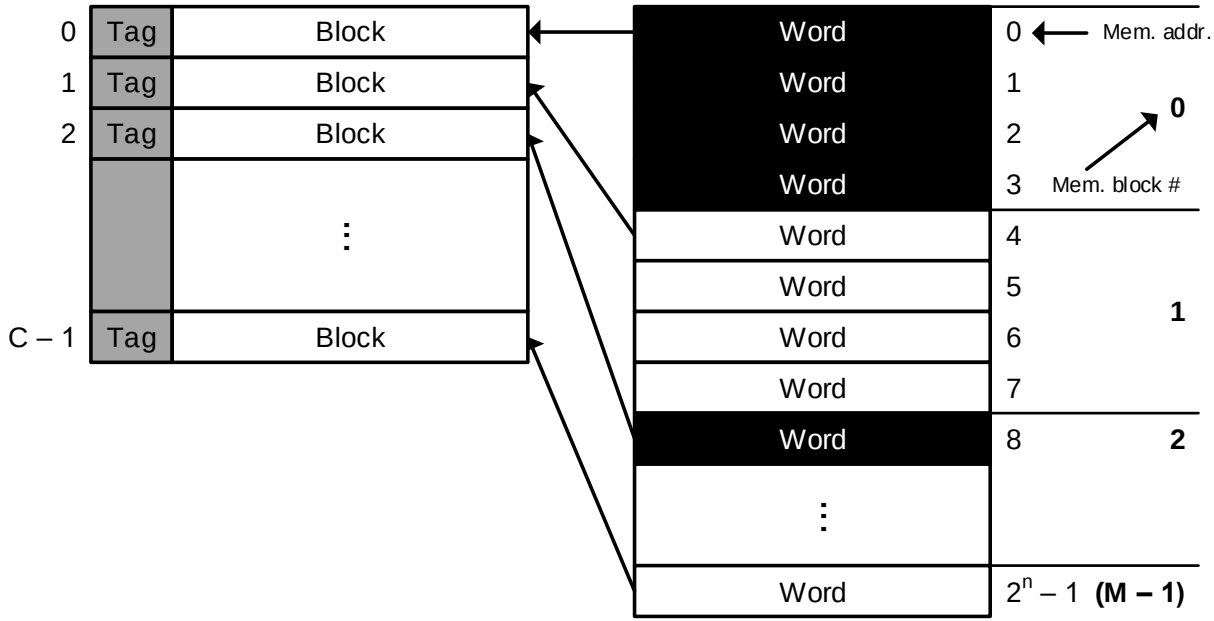
- Outra parte do projeto é a **capacidade**. Não pode ser grande demais (cara e lenta), nem pequena demais;
- Isso complica ainda mais, visto que cada aplicação reage de maneira diferente ao cache. O desempenho da cache varia muito dependendo da carga;
- Não tem um tamanho "ótimo" de memória cache;
- Outra variável é o **algoritmo de mapeamento**. Como já escrito, $C \ll M$. Como os blocos da RAM são mapeados na cache?
- Existem o mapeamento **direto**, **associativo** e **associativo em conjunto**.

Projeto de memória cache – mapeamento direto

- É o **mais simples** de implementar. Cada bloco de memória é mapeado **exclusivamente** em uma linha da memória cache:

$$c = m \bmod C,$$

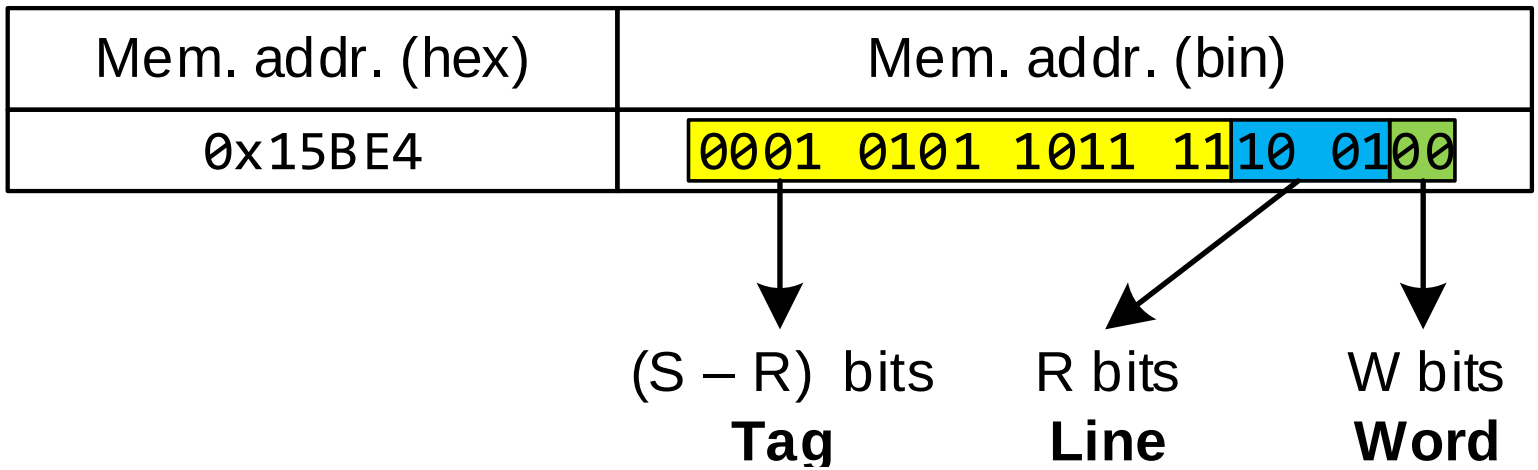
onde c é a linha, $0 \leq m \leq (M-1)$ é o número do bloco e C é o n.º linhas da cache.



- Entretanto, pode haver ***thrashing*** se um programa acessar frequentemente dois blocos mapeados na mesma linha de cache.

Projeto de memória cache – mapeamento direto

Endereço de memória de $N = 20$ bits; cache de dezesseis linhas ($C = 16$; $R = \log_2 C = 4$); blocos de quatro UEs ($K = 4$; $W = \log_2 K = 2$). Os $S = (N - W)$ MSBits do endereço são usados para identificar o bloco, onde os $(S - R)$ MSBits são a *tag*, os W LSBits são usados para identificar a palavra e os R restantes identificam a linha.



Em quais as linhas são mapeados os endereços de memória `0x15BE5`, `0xFBC94`, `0x8942A`, `0x00024`?

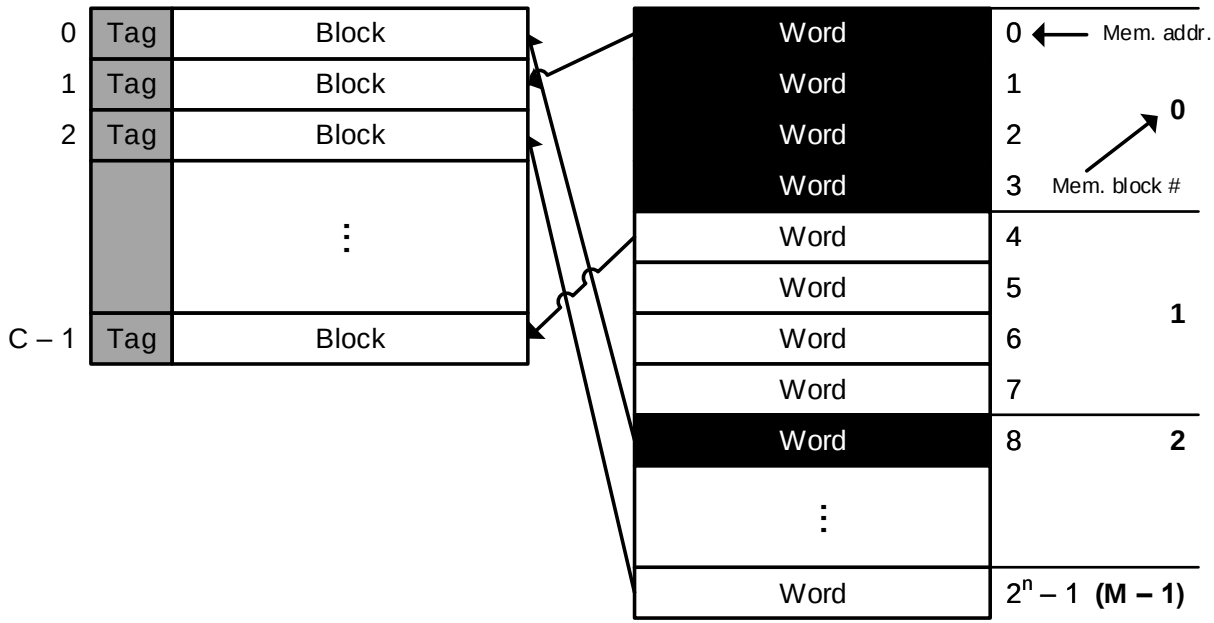
Projeto de memória cache – mapeamento direto

Mapeamento direto dos blocos nas linhas da cache:

Linha da cache	**Endereço inicial dos blocos de memória**
0	$0, 2K, 3K, 4K \cdots$
1	$1, (2K+1), (3K+1), (4K+1) \cdots$
2	$2, (2K+2), (3K+2), (4K+2) \cdots$
3	$3, (2K+3), (3K+3), (4K+3) \cdots$
$\vdots$	$\vdots$
$C-1$	$C-1, (2K+C-1), (3K+C-1) \cdots$

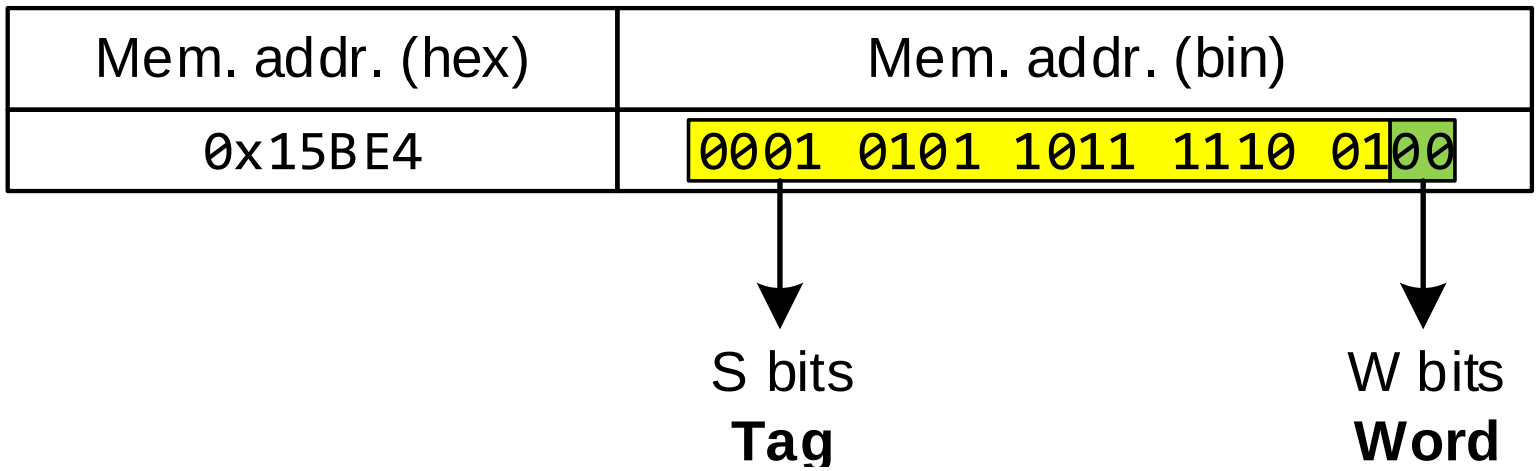
Projeto da memória cache – mapeamento associativo

- Uma cache associativa mapeia um bloco de memória em **qualquer** linha;
- Em vez de usar o campo *Linha* para referenciar bloco em cache, utiliza-se o campo *Tag*;
- O campo *Tag* de todas as linhas é comparado simultaneamente com os S MSBits do endereço de memória acessado;
- Exige um algoritmo de substituição de blocos em cache;



Projeto da memória cache – mapeamento associativo

Endereço de memória de $N = 20$ bits; cache com linhas que são blocos de quatro UEs ($K = 4$; $W = \log_2 K = 2$). Os $S = (N - W)$ MSBits do endereço são usados para identificar o bloco (*Tag*) e os W LSBits são usados para identificar a palavra.

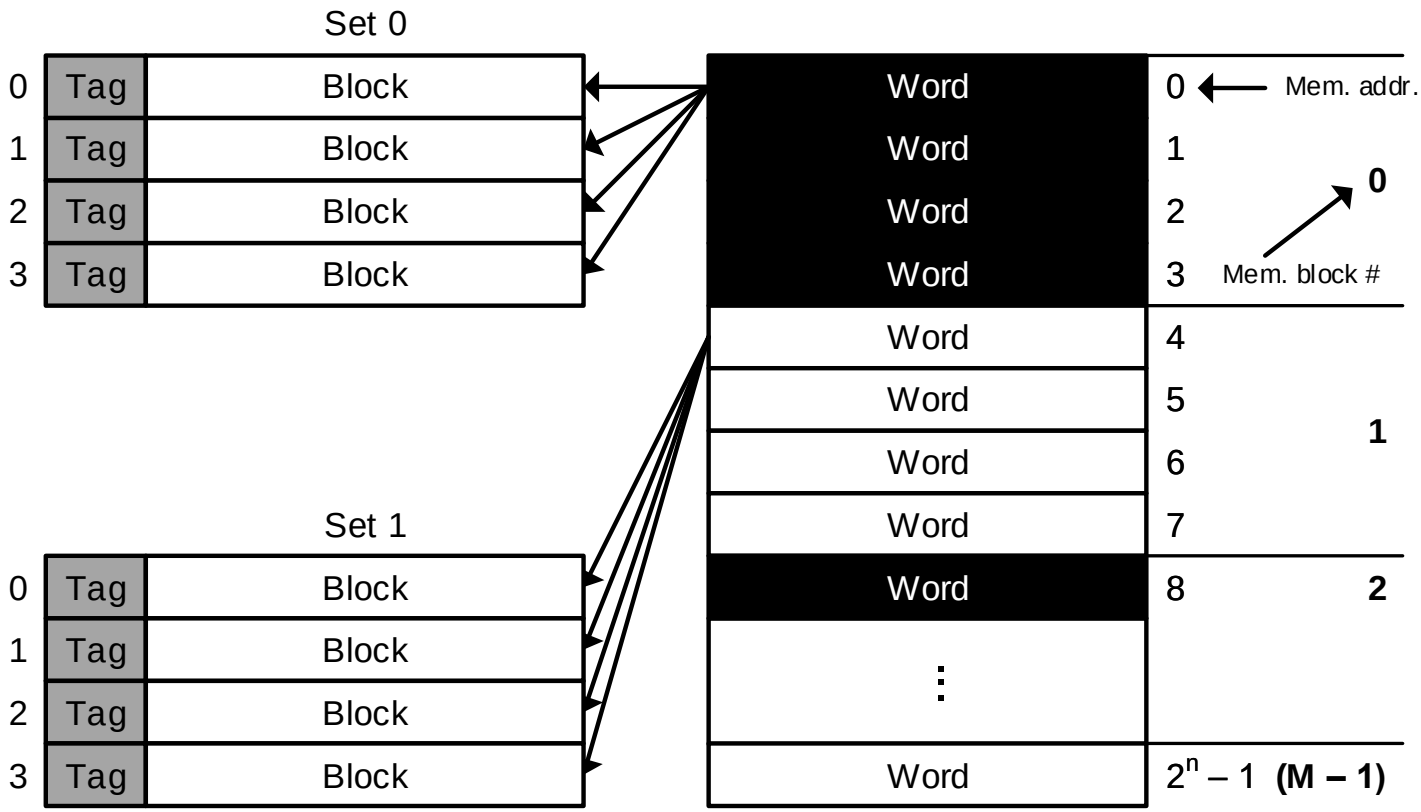


Projeto da memória cache – mapeamento associativo em conjunto

- Quanto maior a cache associativa, mais lenta ela é;
- É um meio-termo entre cache com mapeamento direto e com mapeamento associativo;
- É formado por uma série de T conjuntos de Q linhas (associatividade Q-way) ($C = T \times Q$);
- Pode ser implementado utilizando caches em paralelo com mapeamento direto (para T pequeno) (**como?**) ou associativo (para T grande) (**próx. slide**).

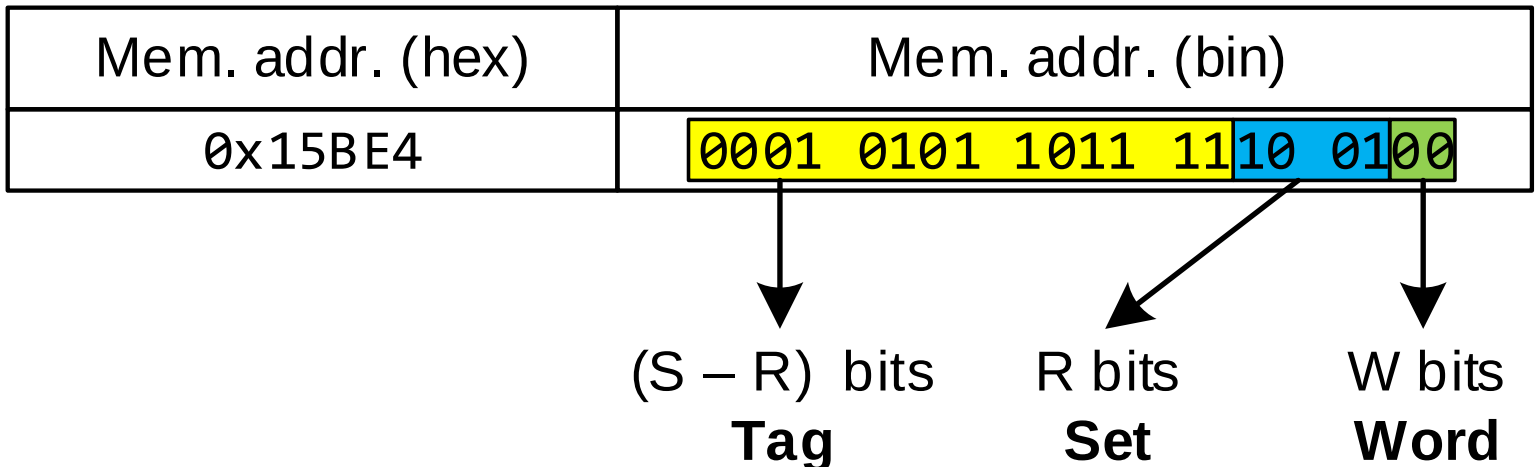
Projeto da memória cache – mapeamento associativo em conjunto

Figura: Cache de oito linhas com mapeamento associativo *4-way*



Projeto da memória cache – mapeamento associativo em conjunto

Endereço de memória de $N = 20$ bits; cache com $C = 128$ linhas e associatividade *8-way* ($Q = 8$). Assim, $T = 128/8 = 16$; $R = \log_2 T = 4$; blocos de quatro UEs ($K = 4$; $W = \log_2 K = 2$). Os $S = (N - W)$ MSBits do endereço são usados para identificar o bloco, onde os $(S - R)$ MSBits são a *tag*, os W LSBits são usados para identificar a palavra enquanto os R restantes identificam o *set*.



Em quais *sets* são mapeados os endereços de memória `0x15BE5`, `0xFBC94`, `0x8942A`, `0x00024`?
E se considerássemos mapeamento direto *8-way*? Como seriam os campos?

Projeto da memória cache

- **Algoritmo de substituição**: no caso de caches associativas, qual linha será *evicted* para dar espaço à outra linha?
  - **LRU** (*Least Recently Used*): a linha que está a mais tempo na cache sem referência nenhuma;
  - **FIFO** (*First-in, first-out*): a linha mais antiga na cache;
  - **LFU** (*Least Frequently Used*): a linha menos usada na cache;
  - **Aleatório**: qualquer linha.
- **Política de escrita**: se um bloco a ser substituído **não** foi alterado, simplesmente é descartado; **senão**, tem que atualizar a memória RAM.
- Mas e no caso de DMA ou várias CPUs? Outros dispositivos acessam a memória: o bloco pode estar inválido;
- A política mais simples é: ***write-through***: memória sempre válida, mas pode ser um gargalo;
- A outra, mais complexa é: ***write-back***: escrita somente na cache $\rightarrow$ memória inválida, mas minimiza o tráfego. Circuitos mais complexos e também pode ser um gargalo.

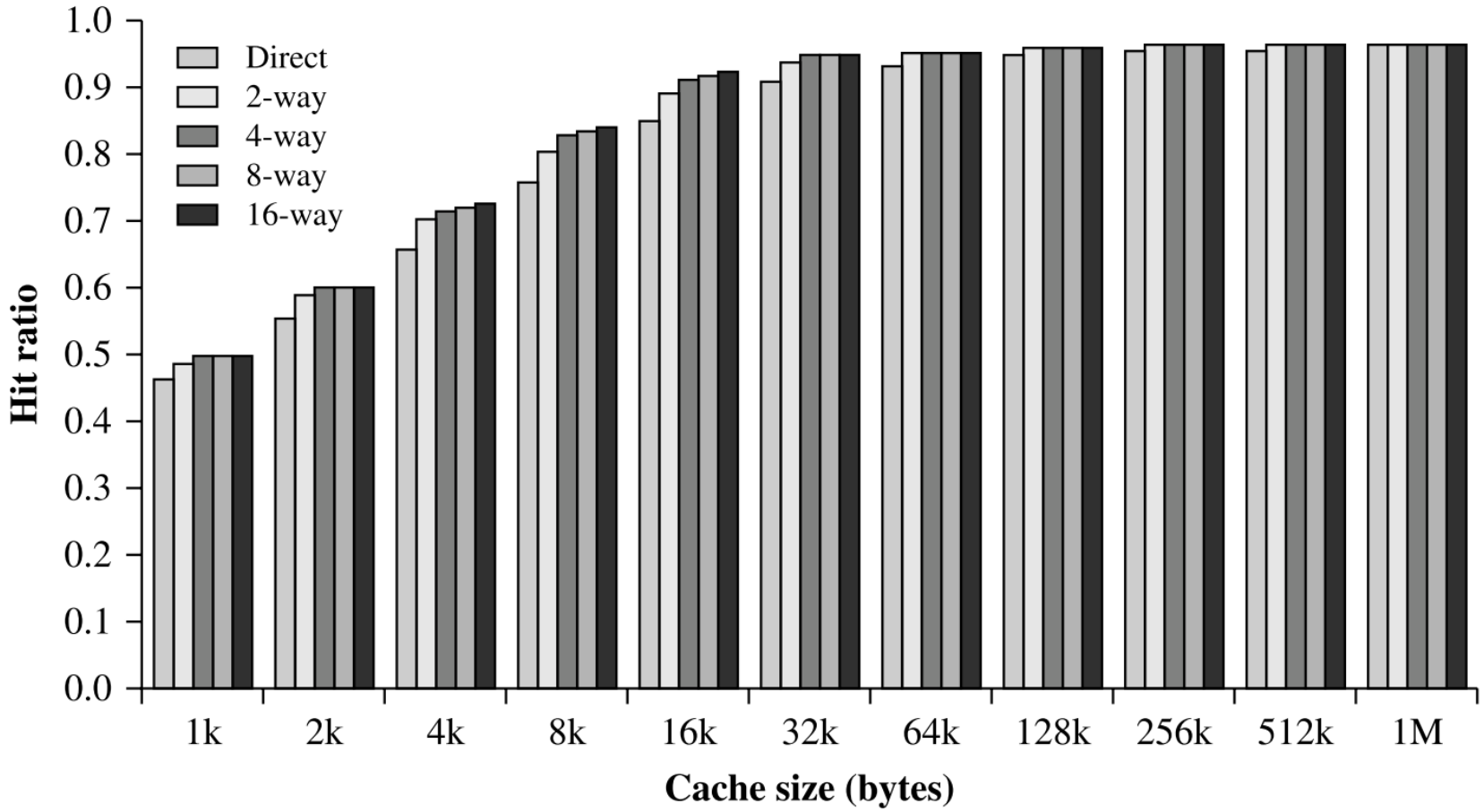
Projeto da memória cache

Política de escrita:

- Circuitos mais complexos $\rightarrow$ **coerência de cache**:
  - Cada cache monitora o tráfego com a RAM e invalida os blocos alterados, mas todas as caches devem usar *write-through*;
  - Uma camada de abstração (um hardware adicional) garante que todas as caches estejam atualizadas;
  - Memória não-cacheável: partes compartilhadas da RAM não são mantidas em cache.
- **Tamanho do bloco**: até certo ponto, quanto maior o bloco, maior a **taxa de acerto** (***cache hit***); por outro lado, menos blocos cabem em uma cache e as palavras do fim do bloco ficam muito distantes e talvez não sejam utilizadas. Normalmente: de oito a 64 bytes;
- **N.º de caches multinível**: com transístores menores surgiram os cache *on die*/*on chip*. Mas compensa ter uma cache fora da CPU? Normalmente sim. Antigamente: on chip L1 (*level*/nível 1) e as outras off chip. L2 do tipo SRAM é mais rápida do que DRAM mesmo estando na placa-mãe;

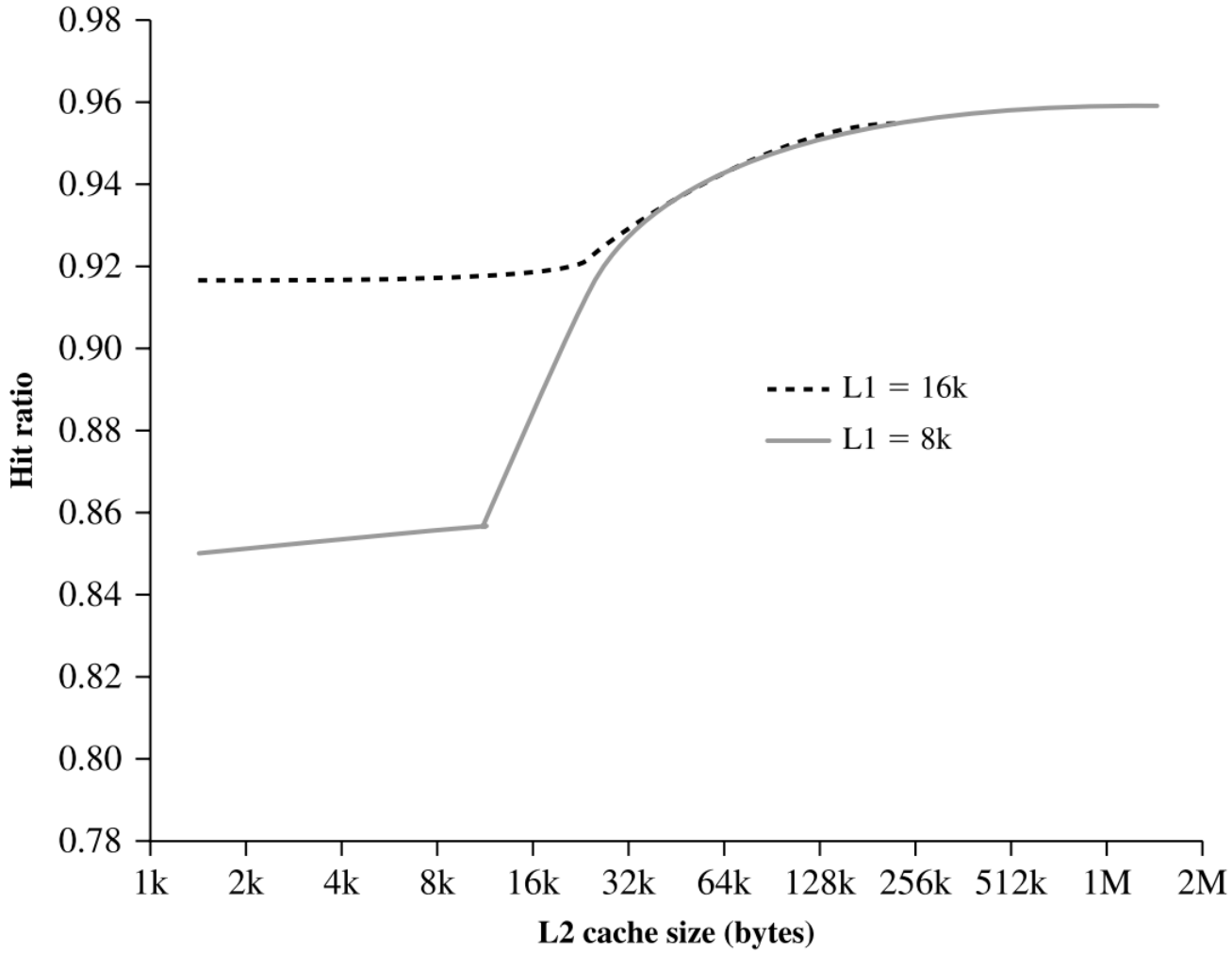
Projeto da memória cache

Figura: Taxa de acerto em função do tamanho da cache e da associatividade



Projeto da memória cache

Figura: Taxa de acerto em função do tamanho das caches L1 e L2



Projeto da memória cache

Caches unificadas vs. caches separadas:

- **Caches unificadas** armazenam **dados** e **instruções**;
- **Caches separadas** armazenam um **ou** outro;
- Normalmente caches L1 são separadas;
- Unidade de busca de instruções (cache de instruções) não compete pelo cache com a unidade de execução (cache de dados);
- É muito importante para CPUs superescalares, superpipelined e as que realizam *pre-fetch*;
- Caches grandes e unificados possuem taxas de acerto maiores pois são mais flexíveis no armazenamento de dados e instruções: L2 e principalmente a LLC (*Last Level Cache*) (L3 ou L4) normalmente são unificadas, ainda mais em CPUs *multicore*.

Exercícios

Capítulo abordado: 4